Datagram Transport Layer Security in Constrained Environments

draft-hartke-core-codtls-01

Klaus Hartke • Olaf Bergmann

Datagram Transport Layer Security in Constrained Environments

- Smart Objects: want the security that DTLS provides
- DTLS has not been designed with constrained devices and low-power, lossy networks in mind
- This is actually not a problem for many constrained devices, but there are some challenges when it comes to implement DTLS for at least Class 1 devices
- *draft-hartke-core-codtls:*
  - trying to figure out challenges and problems
  - collect ideas for possible solutions and implementation guidance

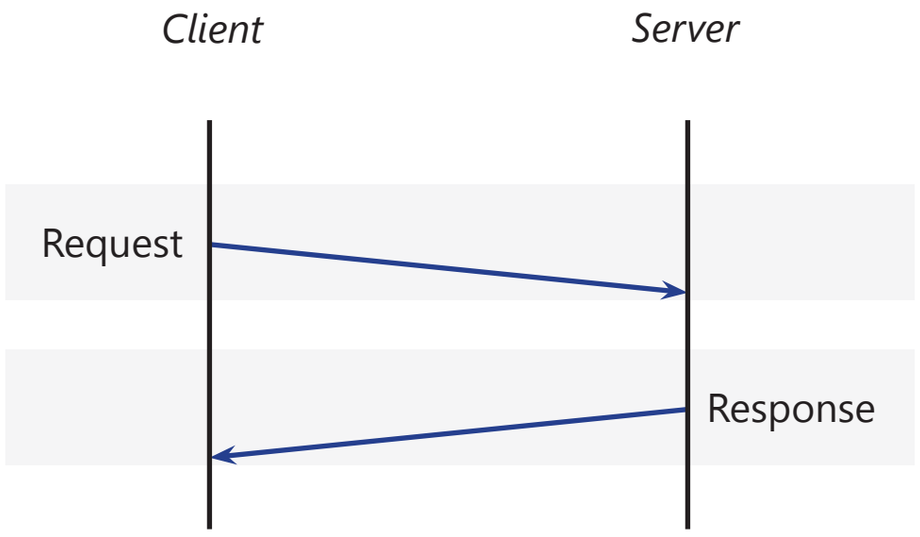
Class 0: too small to securely run on the Internet

Class 1: ~10 KiB data, ~100 KiB code
"quite constrained"

Class 2: ~50 KiB data, ~250 KiB code
"not so constrained"

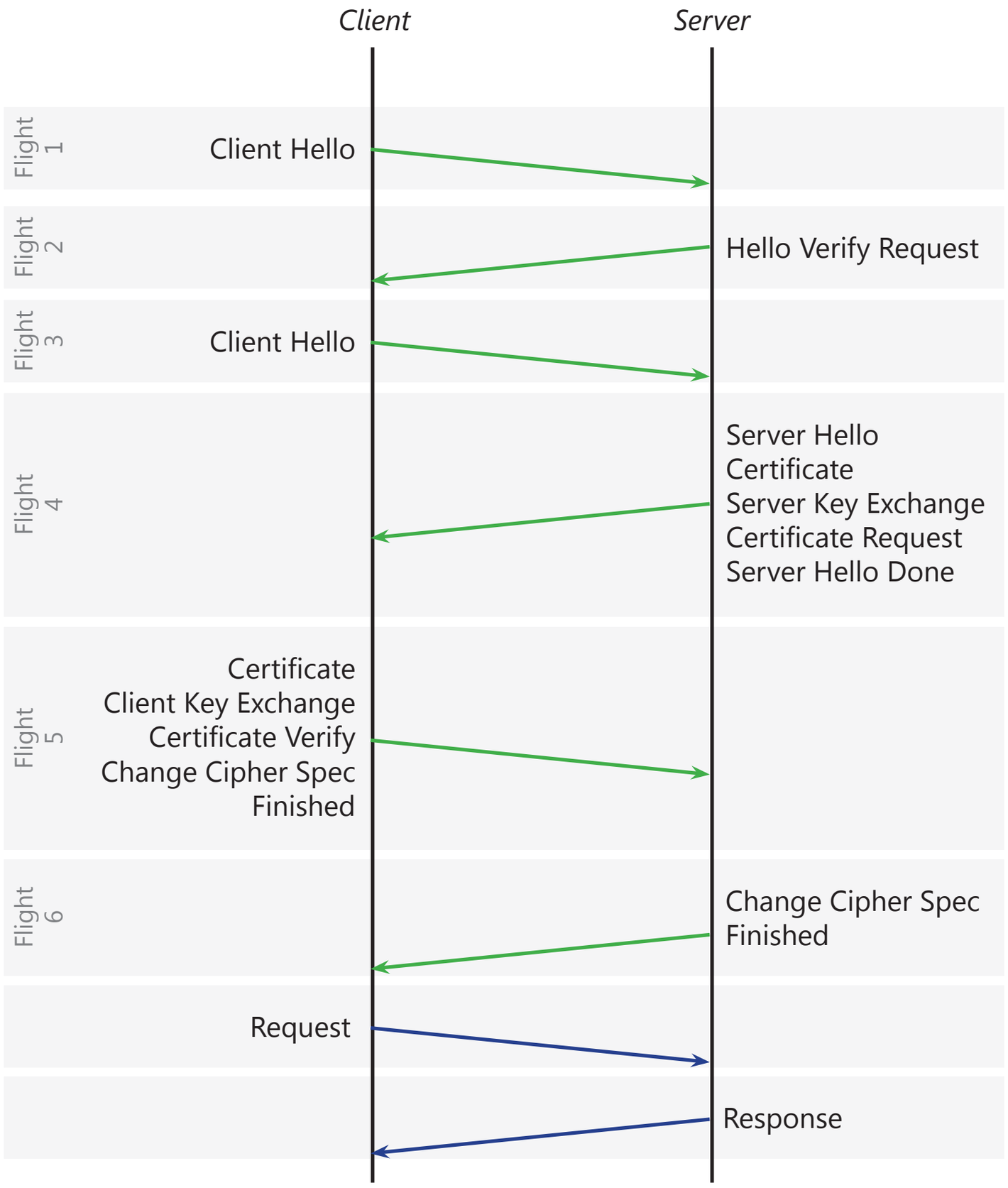
Classes of Devices

CoAP without DTLS

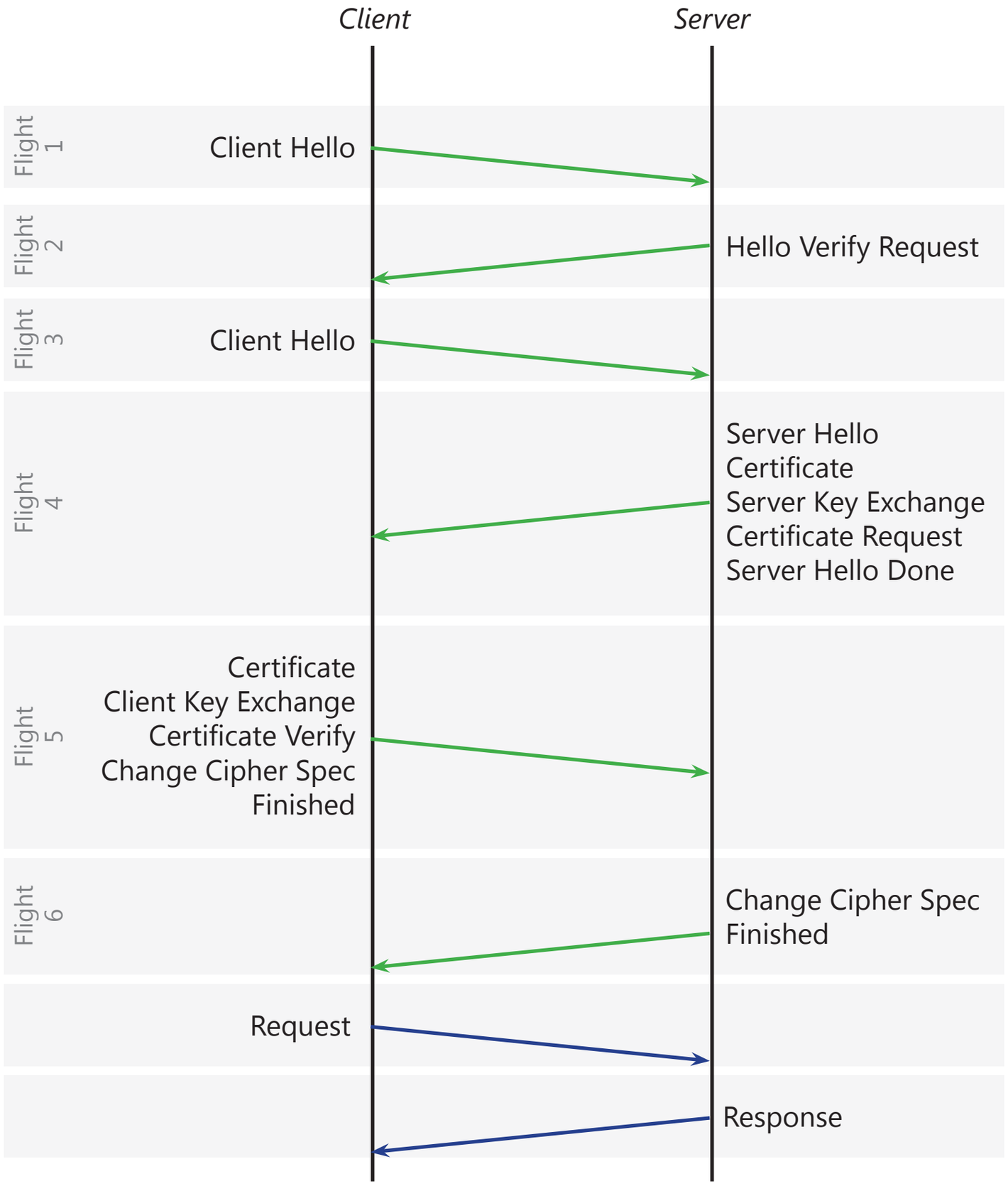


1 roundtrip

CoAP with DTLS



4 roundtrips



4 roundtrips

CoAP with DTLS

DTLS handshake over 6LoWPAN:
max ~ **30-60 bytes** per fragment

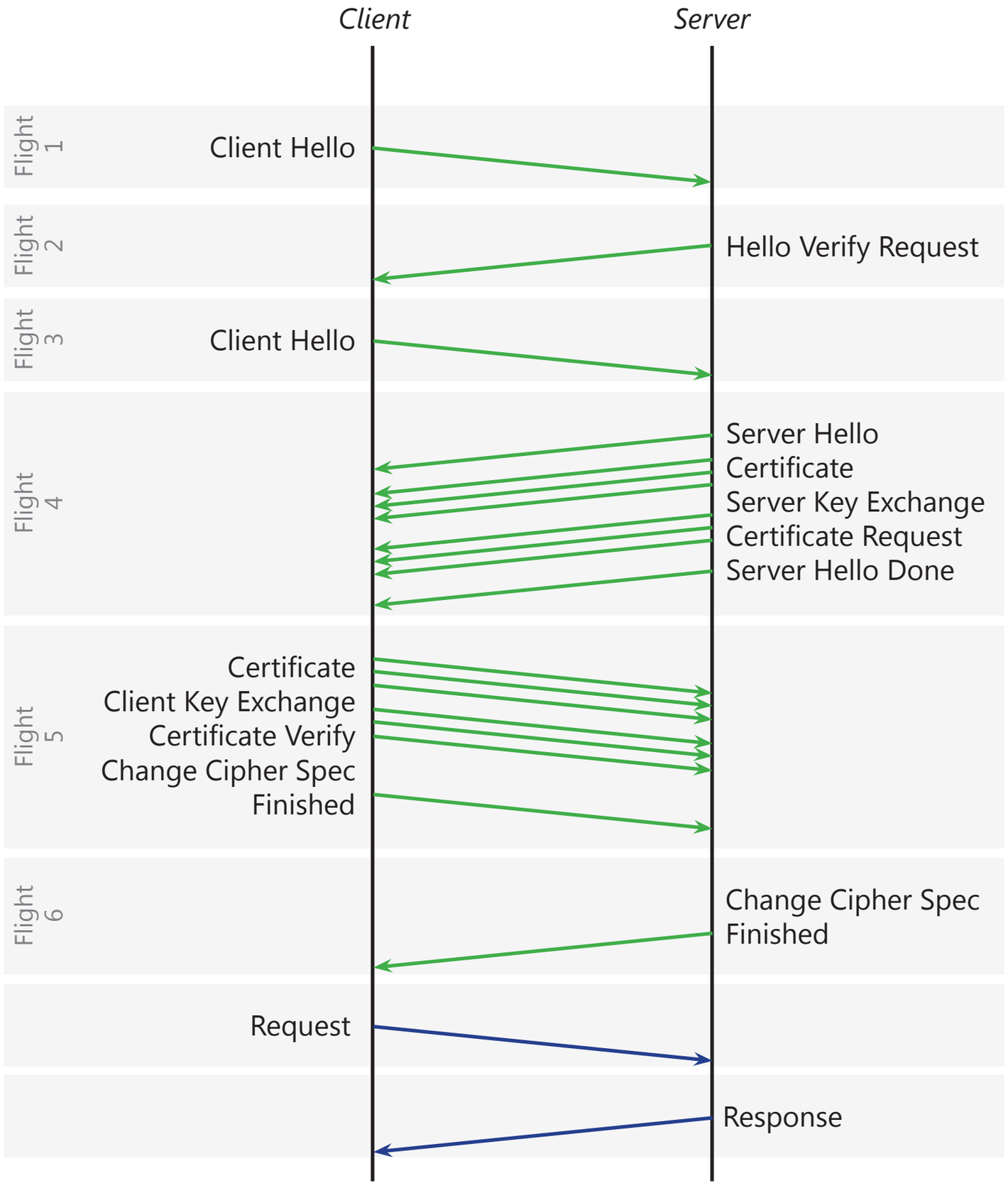
ECDSA P-256: **91 bytes**

ECDSA P-384: **120 bytes**

ECDSA P-521: **156 bytes**

Raw Public Key: Certificate sizes

CoAP with DTLS



4 roundtrips

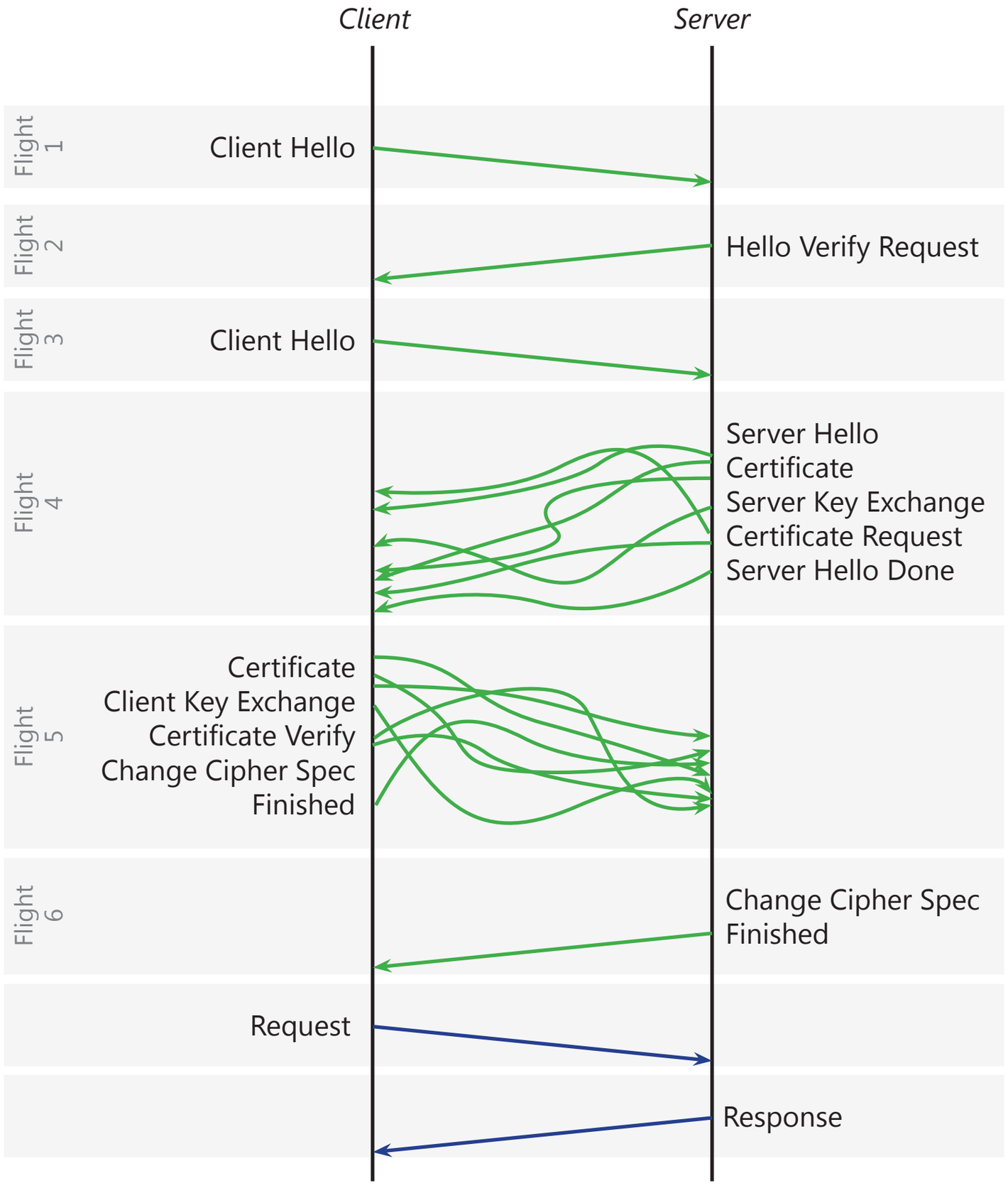
DTLS handshake over 6LoWPAN:
max ~ **30-60 bytes** per fragment

ECDSA P-256: **91 bytes**

ECDSA P-384: **120 bytes**

ECDSA P-521: **156 bytes**

Raw Public Key: Certificate sizes



CoAP with DTLS

DTLS handshake over 6LoWPAN:
max ~ **30-60 bytes** per fragment

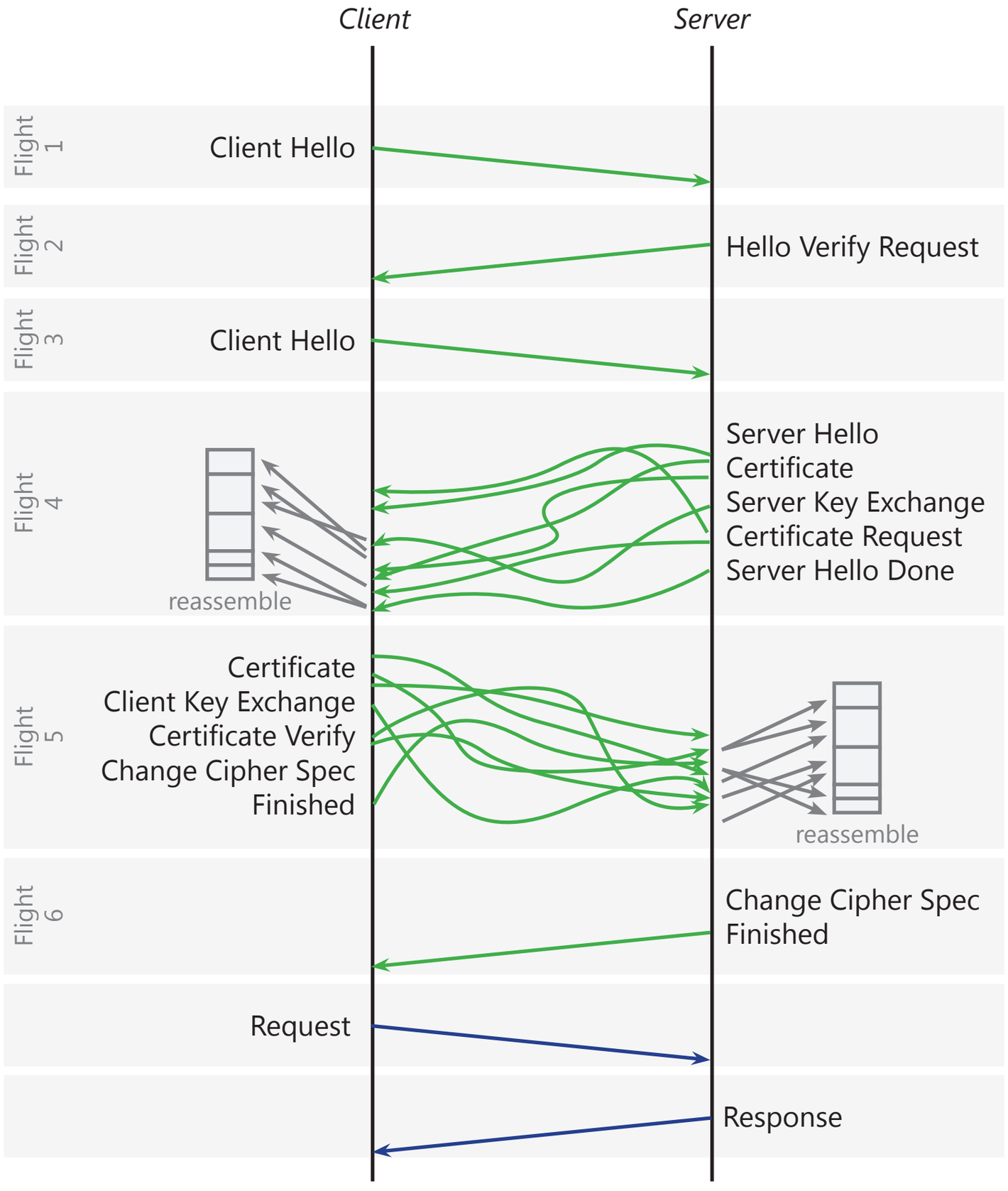
ECDSA P-256: **91 bytes**

ECDSA P-384: **120 bytes**

ECDSA P-521: **156 bytes**

Raw Public Key: Certificate sizes

4 roundtrips



CoAP with DTLS

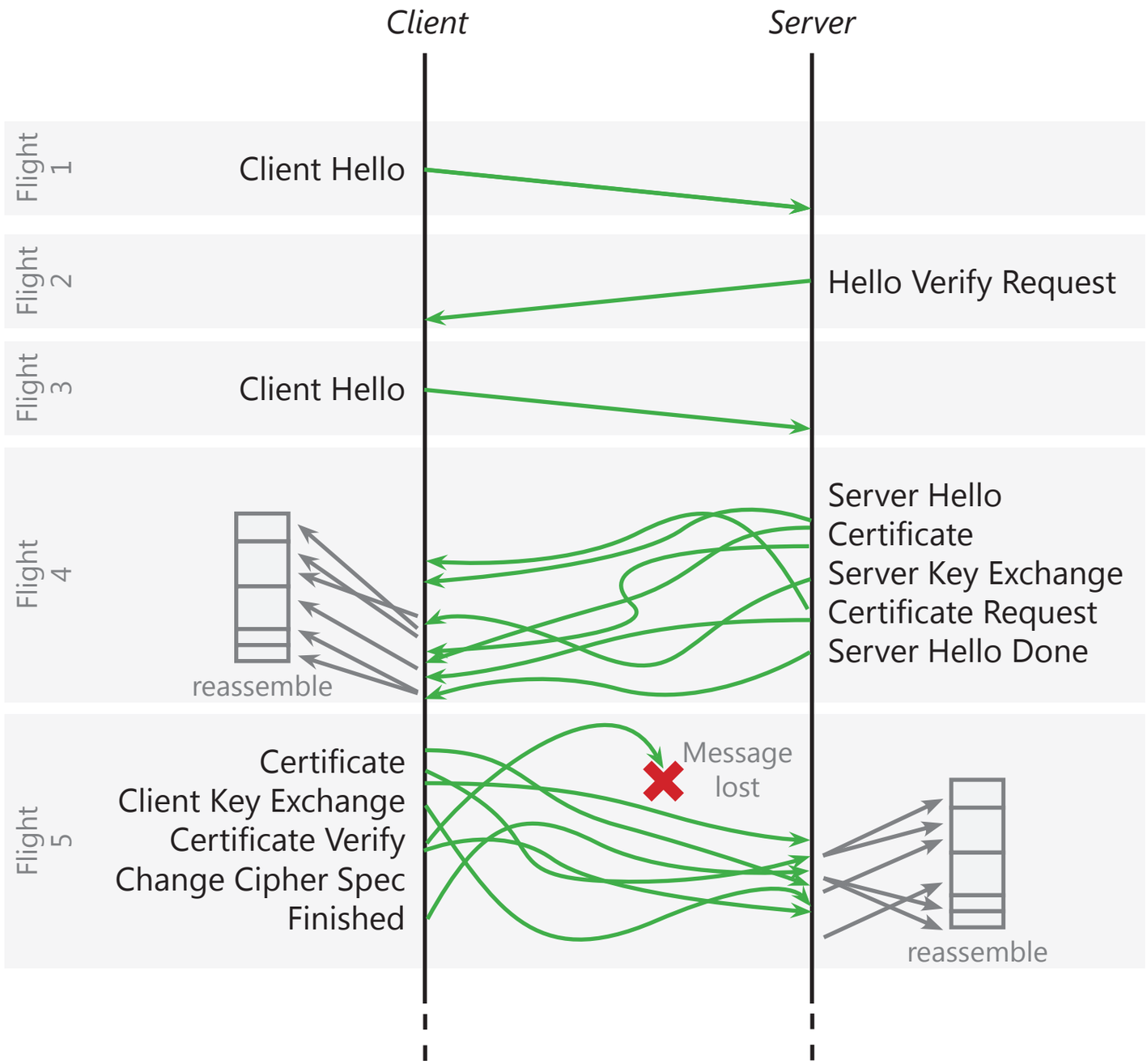
DTLS handshake over 6LoWPAN:
max ~ **30-60 bytes** per fragment

ECDSA P-256: **91 bytes**

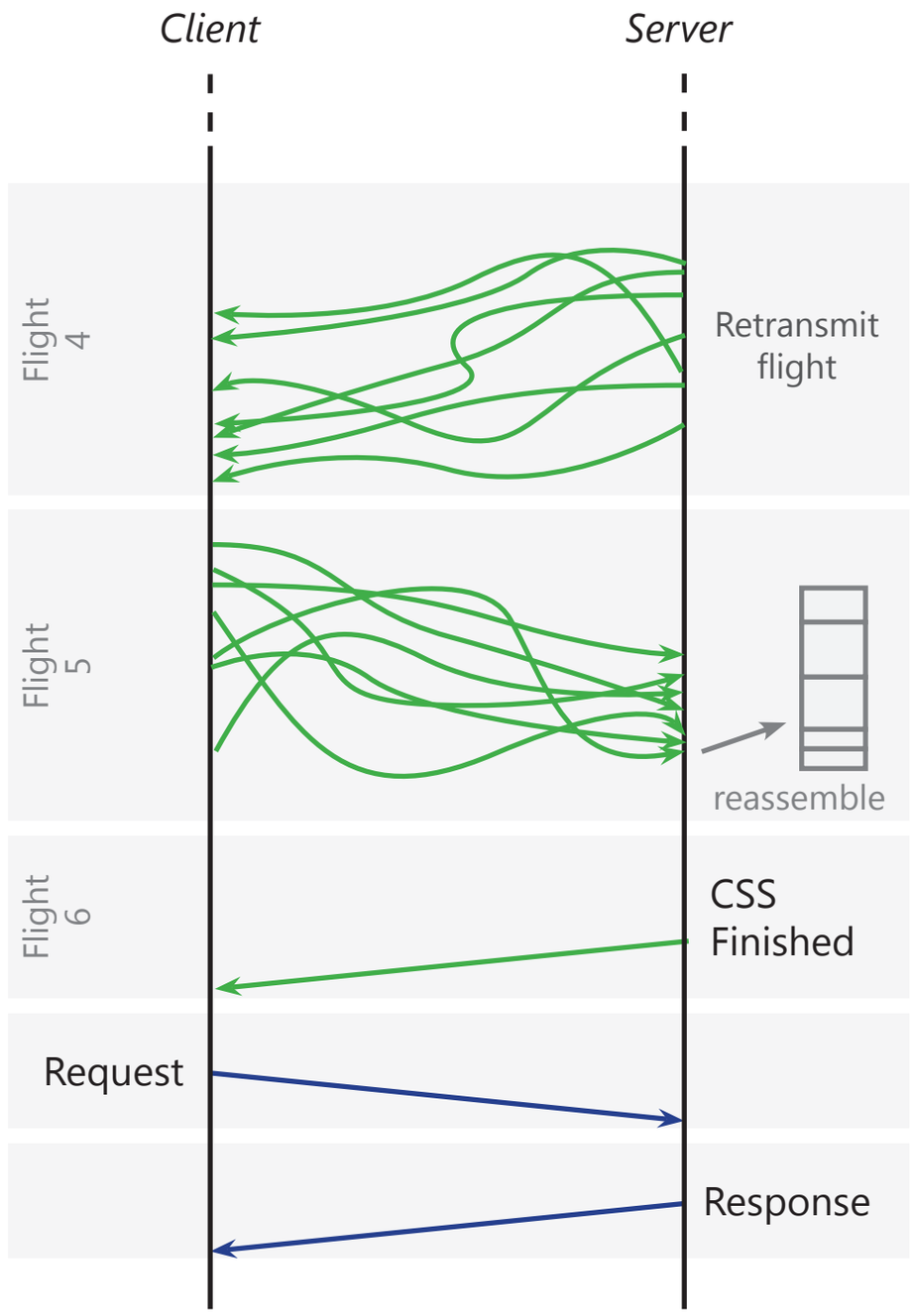
ECDSA P-384: **120 bytes**

ECDSA P-521: **156 bytes**

Raw Public Key: Certificate sizes



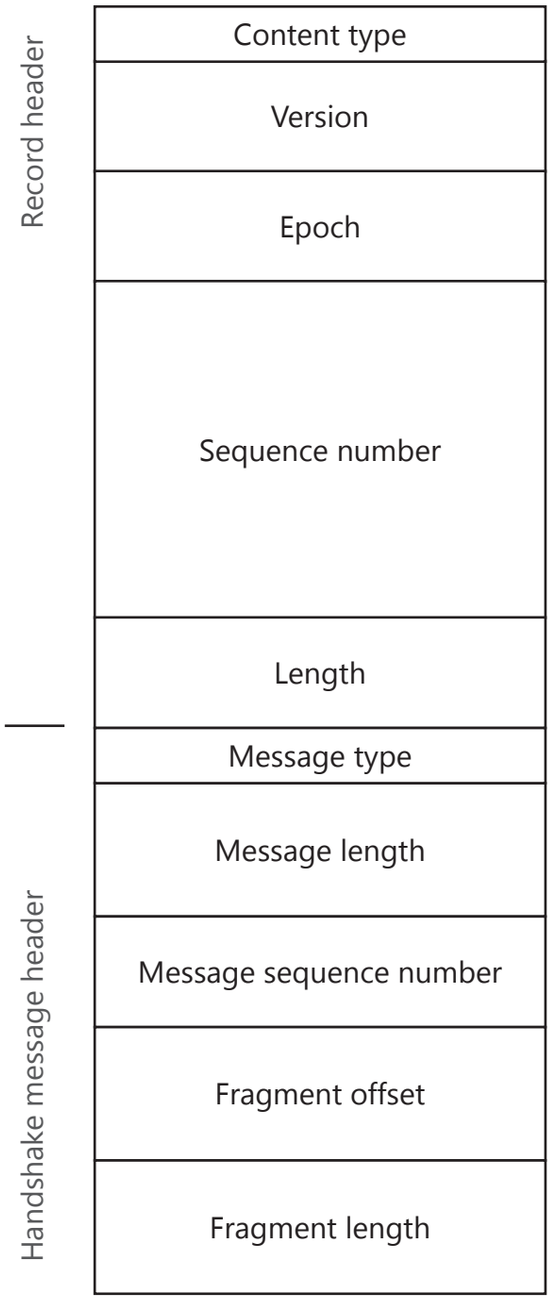
CoAP with DTLS



5 roundtrips

Handshake protocol – Potential problems

- Handshake messages are big
  - We need many frames to transport keys
  - In principle: DTLS fragmentation is better than adaptation layer fragmentation
- DTLS has 25 bytes overhead per packet that carries a fragment
- Fills ~ 1/3 of usable frame
- More fragments increase likelihood that messages get reordered or lost
- Every new packet uses energy



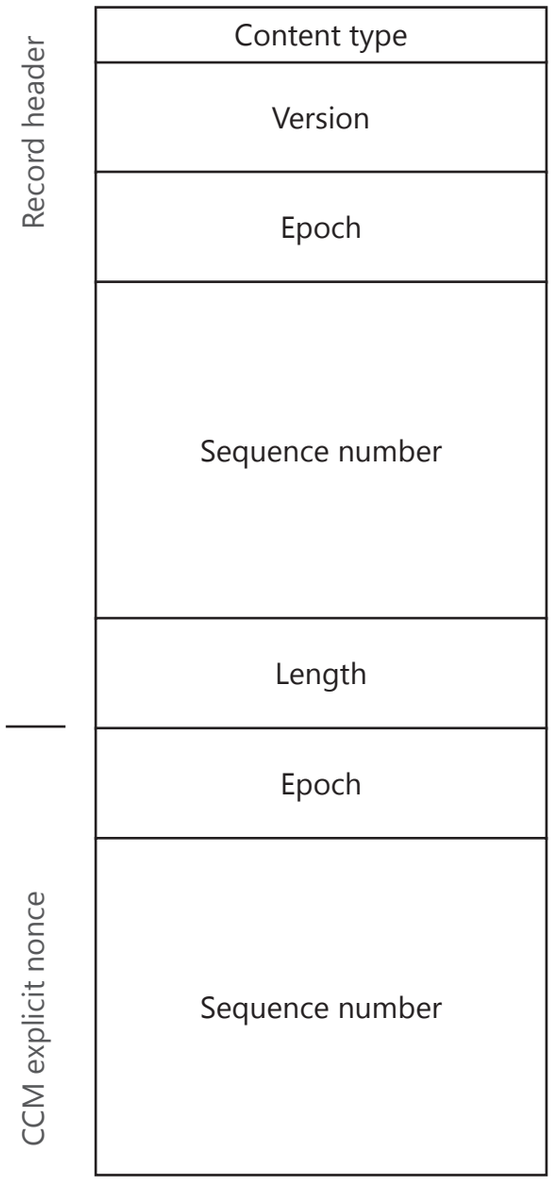
= 25 bytes

Handshake protocol – Possible solutions

- (not to be decided here)
- Compress headers so more data can be transmitted per fragment
  - Can 6LoWPAN GHC handle this?
    - Literal copies are replaced by references
    - Zero sequences can be compressed
  - Or should this better be done end-to-end?
- Is reordering is unlikely enough that a recipient can simply ignore reordered messages and wait for retransmission?

Application data protocol – Potential problems

- DTLS has 21 bytes overhead per packet
  - DTLS version is always the same (2 bytes)
  - Epoch is mostly 0 or 1 (2 bytes)
  - Sequence number is 6 bytes
  - Length is redundant in last record in datagram
  - CCM doubles sequence number and epoch
- Fills ~ 1/3 of usable frame
- Every new packet uses energy



= 21 bytes

Application data protocol – Possible solutions

- (not to be decided here)
- Compress headers so more data can be transmitted per fragment
  - – Can 6LoWPAN GHC handle this?
    - Literal copies are replaced by references
    - Zero sequences can be compressed
  - – Or should this better be done end-to-end?

State –
Potential problems

- Reassembling fragmented handshake messages
- Queuing reordered handshake messages
- Create connection state
  – can often be done per-fragment
- Create verification hash for *Finished* message
  – computed from sequence of messages
  – reordering requires queuing

State –
Possible solutions

- (not to be decided here)
- Possibly compute the hash in the *Finished* message from the negotiated session parameters?

Other guidance

- Adjust the timers...

Algorithm	Library	RAM	Time
RSA-512	AvrCryptolib	320 B	25.0 s
RSA-1024	AvrCryptolib	640 B	199.0 s
ECC 128r1	TinyECC	776 B	1.8 s
ECC 192k1	TinyECC	1008 B	3.4 s
NIST K163	Relic	2804 B	0.3 s
NIST K233	Relic	3675 B	1.8 s

Time to sign (Arduino) [Mohit Sethi]

> Implementations SHOULD use an initial timer value of 1 second (the minimum defined in RFC 6298 [RFC6298]) and double the value at each retransmission, up to no less than the RFC 6298 maximum of 60 seconds. Note that we recommend a 1-second timer rather than the 3-second RFC 6298 default in order to improve latency for time-sensitive applications. Because DTLS only uses re-transmission for handshake and not dataflow, the effect on con-gestion should be minimal.

DTLS Timer Values

Other guidance

- Data size…
- Code size…

Algorithm	Library	RAM	Time
RSA-512	AvrCryptolib	320 B	25.0 s
RSA-1024	AvrCryptolib	640 B	199.0 s
ECC 128r1	TinyECC	776 B	1.8 s
ECC 192k1	TinyECC	1008 B	3.4 s
NIST K163	Relic	2804 B	0.3 s
NIST K233	Relic	3675 B	1.8 s

RAM usage (Arduino) [Mohit Sethi]

Class 0: too small to securely run on the Internet

Class 1: ~10 KiB data, ~100 KiB code
"quite constrained"

Class 2: ~50 KiB data, ~250 KiB code
"not so constrained"

Classes of Devices

Code Size	Description
1429 Bytes	SHA-256
992 Bytes	CCM
9812 Bytes	DTLS state machine

Code footprint of minimal DTLS implementation [Olaf Bergmann]

Possible actions

prefer

Implementation guidance
Implementation techniques for light-weight implementations without affecting conformance

→ I-D.bormann-lwig-guidance

Stateless header compression
Compress record and handshake headers without explicitly building any compression context state

Protocol profile for constrained environments
Use of DTLS in a particular way, e.g.
- require or preclude certain extensions or cipher suites
- change MAYs into MUSTs or MUST NOTs

→ CoRE WG (?)

avoid

Breaking changes

→ TLS WG